A point-in-polygon method based on a quasi-closest point

Sheng Yang, Jun-Hai Yong, Jiaguang Sun, He-Jin Gu, Jean-Claude Paul

A point-in-polygon method based on a quasi-closest point

Sheng Yang [a,b,c,d,e,*], Jun-Hai Yong [a,d], Jiaguang Sun [a,d], Hejin Gu [a,f], Jean-Claude Paul [a,g]

[a] Institute of CG & CAD, School of Software, Tsinghua University, Beijing 100084, China
[b] Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[c] Third Military Medical University, Chongqing 400038, China
[d] Key Laboratory for Information System Security, Ministry of Education, Beijing 100084, China
[e] Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China
[f] Jiangxi Academy of Sciences, Nanchang 330029, China
[g] INRIA, France

ARTICLE INFO

Article history:
Received 3 October 2008
Received in revised form
22 May 2009
Accepted 20 June 2009

Keywords:
Point-in-polygon
Uniform subdivision
Closest point
Robustness

ABSTRACT

This paper presents a numerically stable solution to a point-in-polygon problem by combining the orientation method and the uniform subdivision technique. We define first a quasi-closest point that can be locally found through the uniform subdivision cells, and then we provide the criteria for determining whether a point lies inside a polygon according to the quasi-closest point. For a large number of points to be tested against the same polygon, the criteria are employed to determine the inclusion property of an empty cell as well as a test point. The experimental tests show that the new method resolves the singularity of a test point on an edge without loss of efficiency. The GIS case study also demonstrates the capability of the method to identify which region contains a test point in a map.

1. Introduction

Point-in-polygon test (also called inclusion test), which determines whether a point lies inside or outside a polygon, is one of the most fundamental operations in computational geometry (Preparata and Shamos, 1991). It can be used in many applications such as computer graphics and geographic information systems (GIS). For example, every time a mouse is clicked inside a map on a screen, an instance of the point-in-polygon problem occurs (O'Rourke, 2000). Another example in geosciences is to identify if a water well is in a certain watershed, and even to assign regions to hydrologic units for the prediction of erosion and deposition zones (Jiménez et al., 2009). The point-in-polygon test can be also used to determine the number of soil boreholes located in a particular cadastral parcel (Ellis, 2001). In particular, the test is the basis for other algorithms such as polygon clipping, ray tracing solids and point-in-polyhedron test. Therefore, it is necessary to find an efficient and reliable solution to the problem (Huang and Shih, 1997).

According to their requirements for preprocessing, the existing algorithms for the point-in-polygon test can be classified into two categories. In the category without preprocessing, an inclusion property is determined by a certain parameter such as the ray crossing number (Preparata and Shamos, 1991; O'Rourke, 2000), the sum of angles (Hormann and Agathos, 2001), the sign of offset (Taylor, 1994), or the orientation of a point against polygon edges (Nordbeck and Rystedt, 1967). Through analyzing the complexity of these algorithms, Huang and Shih (1997) reported that the ray crossing method is suitable for possibly concave polygons whereas the orientation method is the most efficient and suitable algorithm for convex polygons.

The algorithms with preprocessing emerge to test against the same polygon for a large number of points. These algorithms still obtain an inclusion property from a certain parameter, but evaluating the parameter is accelerated by preprocessing the polygon. Generally, the preprocessing step decomposes the polygon of interest into simpler components such as grids (Huang and Shih, 1997), trapezoids (Salomon, 1978), triangles (Feito et al., 1995; Martinez et al. 2006), convex polygons (Li et al., 2007), star-shaped polygons (Etzion and Rappoport, 1997), uniform-sized cells (Zalik and Kolingerova, 2001; Gombosi and Zalik, 2005), or layer edges (Wang et al., 2005). Thus an inclusion query can be efficiently accomplished by the neighboring components of a test point.

As one of the most popular preprocessing methods, the uniform subdivision has been used by many algorithms, for example, ray tracing (Fujimoto et al., 1986; Cleary and Wyvill, 1988) and closest point query (Preparata and Shamos, 1991). Based on the uniform subdivision, Zalik and Kolingerova (2001)

* Corresponding author at: Institute of CG & CAD, School of Software, Tsinghua University, Beijing 100084, China.
E-mail address: yangsheng05@mails.tsinghua.edu.cn (S. Yang).

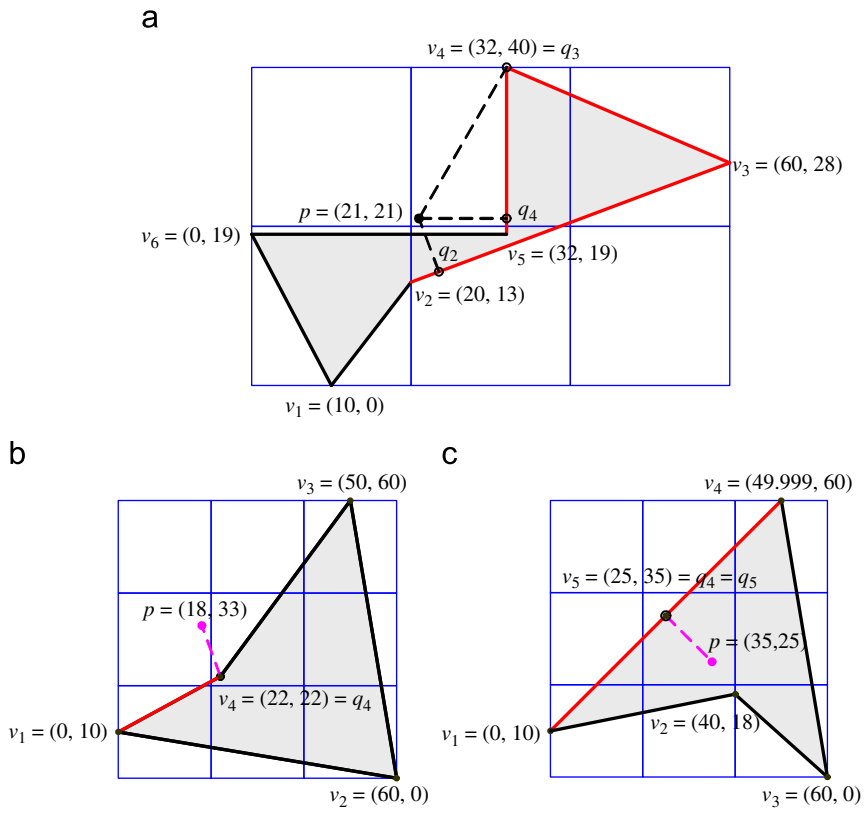

Fig. 1. Three examples of a test point and a polygon with subdivision dimension 20×20.

Table 1
Process of CBCA for test point in Fig. 1.

Example	The cell containing p	The edges crossing the cell	The closest point	Test result
Fig. 1a	Cell(2, 2)	v_2v_3, v_3v_4, v_4v_5	q_2	In
Fig. 1b	Cell(2, 1)	v_4v_1	$q_4{=}v_4$	In
Fig. 1c	Cell(2, 2)	v_4v_5, v_5v_1	$q_4{=}q_5{=}v_5$	In or Out

Note: the position of a cell is represented by Cell(row, column), and the edges are directed.

presented an efficient algorithm for the point-in-polygon problem. In some cases, however, the algorithm can give incorrect answers (see Fig. 1 and Table 1). The purpose of this paper is to present the criteria for the point-in-polygon tests against possibly concave polygons, and moreover, to resolve the singularity of a test point on an edge within a user-defined tolerance.

2. Related work and our improvements

To efficiently obtain the inclusion property, the algorithm named CBCA in (Zalik and Kolingerova, 2001) preprocesses a polygon by subdividing its bounding rectangle into uniform-sized cells. In addition, the preprocessing steps include recording each edge of the polygon into the cells crossed by the edge and classifying the empty cells as inside or outside the polygon. Let p be a test point and $Cell(p)$ be the cell containing p. If $Cell(p)$ is empty, the inclusion property of p is obtained directly from that of $Cell(p)$; otherwise CBCA employs the orientation method to determine the inclusion property, this is, it finds the closest edge (or vertex) of p to the edges stored in $Cell(p)$, and then determines the inclusion property from the orientation of p with respect to the closest edge (or the convexity property of the closest vertex). As shown in Fig. 1 and Table 1, CBCA may produce incorrect result when the closest point is out of $Cell(p)$ or the adjacent edges of the vertex are nearly collinear.

In Fig. 1a, q_2 is not the closest point of p to all the edges, and more importantly, the line segment pq_2 can intersect v_5v_6 so that the orientation of p with respect to v_2v_3 is unrelated to the inclusion property of p. In Fig. 1b, q_4 is not the closest point of p to the edges crossing $Cell(v_4)$, so the convexity property of v_4 is unrelated to the inclusion property of p as well. In Fig. 1c, v_5 is the closest point in the direction from p to v_5, but it is difficult to determine the convexity property of v_5 in the presence of rounding errors; especially, v_5 is not the closest point of p to all the edges because at least the distance between p and v_2 is smaller than that between p and v_5. It can be seen that finding the closest point in any direction is more expensive than in a certain direction. Underlying these observations are the needs for finding such a parameter that it is not only suitable for the orientation method but also easy to be evaluated through the neighboring cells of a test point, and avoiding the use of the convexity property of a vertex when its adjacent edges are nearly collinear.

In this paper, we define a quasi-closest point to represent the closest point in a certain direction, and then present the criteria for obtaining an inclusion property from the quasi-closest point without using the convexity property of a vertex when its adjacent edges are nearly collinear. In Section 3, the definition and criteria are given in a mathematical way so that the orientation method can be applied to the point-in-polygon tests against possibly concave polygons. Based on the uniform subdivision method, Section 4 presents the algorithm for obtaining the quasi-closest point. Especially, if the point obtained by the algorithm is within the neighborhood of a test point for a user-defined tolerance, it is the closest point in any direction. By the minimum distance, the algorithm for an inclusion query does deal with the singular case that a test point is on an edge. Section 5 illustrates the preprocessing steps of point-in-polygon tests for a large number of test points, where the criteria are employed to obtain the inclusion property of an empty cell. The experimental tests as well as the case study in GIS are demonstrated in Section 6. These results exhibit the characteristics of the new method, for example,

- it works as efficiently as other similar algorithms do;
- it handles the singular case that a point is on an edge; and
- it can be applied to GIS operations such as point-in-polygon overlay.

3. The criteria for point-in-polygon test based on a quasi-closest point

Following the rule in (Zalik and Kolingerova, 2001), we assume that the polygons are simple ones and oriented, e.g. counterclockwise. For a simple polygon, the line segment is called edge, and the point where each pair of adjacent segments intersect is called vertex (O'Rourke, 2000).

3.1. A quasi-closest point

Denote by $d(p_1, p_2)$ the distance between two points p_1 and p_2. Let p be a point and e_i $(1 \le i \le n)$ be an edge of a polygon P. Then the distance from p to e_i is defined by

$$\mathrm{d}(p, e_i) = \min\{d(p, q) | q \in e_i\}.$$

If $q \in e_i$ and $d(p, q) = d(p, e_i)$, q is called the closest point from p to e_i and denoted by $Q(p, e_i)$.

The closest point from p to e_i, say $q = Q(p, e_i)$, is called *a quasi-closest point* from p to P if for the line segment pq,

(i) $pq \cap e_k = \emptyset$ for $1 \le k \le n$, $k \ne i$; or
(ii) $pq \cap e_k = \emptyset$ for $1 \le k \le n$, $k \ne i, j$, and $q = Q(p, e_j)$.

In Fig. 2, q_1 and q_4 are quasi-closest points for they meet the condition (i), whereas q_5 and q_6 are not; q_2 is a quasi-closest point for it meets the condition (ii), whereas q_3 is not. Note that for a test point, there may be several quasi-closest points and one of them is the closest point to all the edges. For convenience, the quasi-closest point meeting the condition (i) is denoted by $Q(p, e_i, P)$, and the one meeting the condition (ii) by $Q(p, e_i, e_j, P)$, where e_i and e_j are adjacent edges (since P is a simple polygon) and the order of e_i and e_j agrees with the direction of P.

3.2. Left/right side

For a triangle $\triangle v_0v_1v_2$, its signed area can be computed as

$$\mathrm{A}(\triangle v_0v_1v_2) = ((x_1 - x_0)(y_2 - y_1) - (y_1 - y_0)(x_2 - x_1))/2,$$

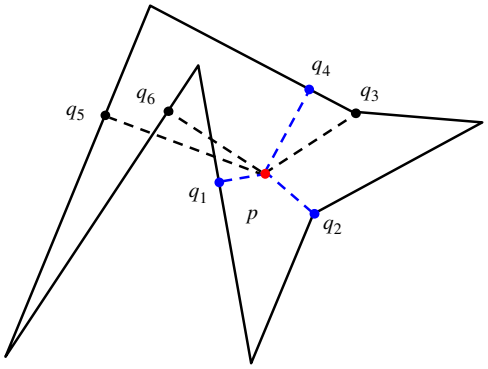

Fig. 2. Closest points from a point to edges of a polygon. Among them, q_1, q_2, and q_4 are quasi-closest points.

where $v_i = (x_i, y_i)$ for $i = 0,1,2$. The signed area possesses the following properties:

(i) If $A(\triangle v_0v_1v_2) > 0$, then $A(\triangle v_1v_2v_0) > 0$.
(ii) If $A(\triangle v_0v_1v_2) > 0$, then $A(\triangle v_1v_0v_2) < 0$.
(iii) If $A(\triangle v_0v_1v_2) > 0$, then $A(\triangle v_0v_1p) > 0$, where $p \in v_1v_2 \backslash \{v_1\}$.

For a directed edge $e_0 = v_0v_1$ and a point p, the point is on the left side of the edge if $A(\triangle v_0v_1p) > 0$, while the point is on the right side of the edge if $A(\triangle v_0v_1p) < 0$. For e_0 and its adjacent edge $e_1 = v_1v_2$, e_1 is on the left side of e_0 if $A(\triangle v_0v_1v_2) > 0$, while e_1 is on the right side of e_0 if $A(\triangle v_0v_1v_2) < 0$.

Denoted by $\angle v_0v_1v_2$ the smaller angle formed by v_0v_1 and v_1v_2. Obviously, $0 < \angle v_0v_1v_2 \le \pi$. For $v_1 = Q(p, e_0, e_1, P)$, the relationship between $\angle v_0v_1v_2$ and the orientation of p with respect to e_0 and e_1 can be stated as follows:

Theorem 1. Let $v_1 = Q(p, e_0, e_1, P)$ and $p \ne v_1$ (see Fig. 3). If $\angle v_0v_1v_2 > \pi/2$, then p is on the same side of e_0 and e_1.

Proof. We wish to show that v_0 and v_2 are on different sides of v_1p. For $v_1 = Q(p, e_0, e_1, P)$ and $p \ne v_1$, we have $\angle v_0v_1p \ge \pi/2$ and $\angle pv_1v_2 \ge \pi/2$. Suppose that $\angle v_0v_1p = \pi$. It follows that $\angle v_0v_1p = \angle v_0v_1v_2 + \angle pv_1v_2 > \pi$, contrary to the hypothesis. Therefore, $\pi/2 \le \angle v_0v_1p < \pi$; by a similar proof, $\pi/2 \le \angle pv_1v_2 < \pi$. If v_0 and v_2 are on the same sides of v_1p, it follows that $\angle v_0v_1v_2 < \pi/2$, which contradicts the statement that $\angle v_0v_1v_2 > \pi/2$. So there are two cases:

(1) If $A(\triangle v_1pv_0) > 0$ and $A(\triangle v_1pv_2) < 0$, it follows that $A(\triangle v_0v_1p) > 0$ due to the property (i), and $A(\triangle v_1v_2p) > 0$ due to the property (ii). This means that p is on the left side of both e_0 and e_1.

(2) If $A(\triangle v_1pv_0) < 0$ and $A(\triangle v_1pv_2) > 0$, it follows that $A(\triangle v_0v_1p) < 0$ and $A(\triangle v_1v_2p) < 0$. This means that p is on the right side of both e_0 and e_1. □

3.3. Point-in-polygon test based on a quasi-closest point

Theorem 2. Let p be a point, P be a simple polygon, and q be a quasi-closest point from p to P. Assume that P is oriented counterclockwise and $p \ne q$. Then p is inside P if one of the following conditions holds:

(i) p is on the left side of e_i for $q = Q(p, e_i, P)$.
(ii) p is on the left side of both e_i and e_j for $q = Q(p, e_i, e_j, P)$.
(iii) e_j is on the right side of e_i for $q = Q(p, e_i, e_j, P)$.

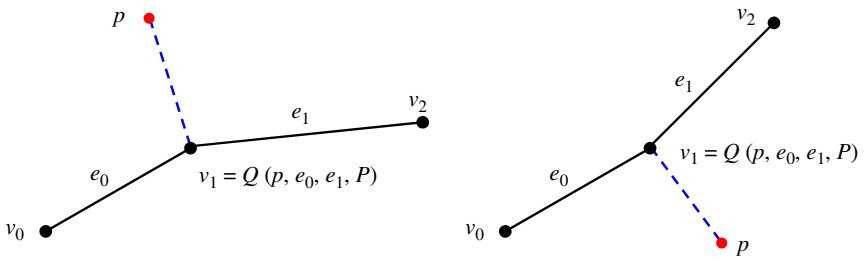

Fig. 3. Illustration for the proof of Theorem 1 (p is on same side of e_0 and e_1 for $\angle v_0v_1v_2 > \pi/2$).

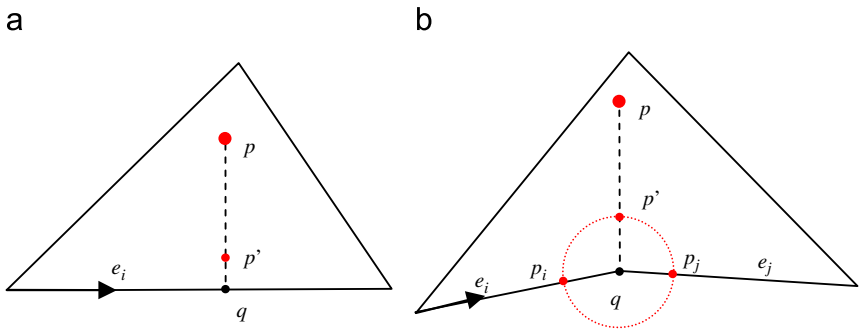

Fig. 4. Illustration for the proof of Theorem 2. (a) $q=Q(p, e_i, P)$; (b) $q=Q(p, e_i, e_j, P)$.

Proof. Since $p \neq q$, we can choose a point $p' \in pq\backslash\{p, q\}$ such that p' is close enough to q (see Fig. 4). Indeed, by the definition of quasi-closest point, $pp' \cap e_k = \emptyset$ $(1 \leq k \leq n)$. If p' is inside P, then p is inside P due to the property of Jordan closed curve.

(i) From the property (iii) of the signed area, since $q=Q(p, e_i, P)$ and p is on the left side of e_i, p' is on the left side of e_i. As a result, p' is inside P by noting that P is oriented counterclockwise and p' is close enough to q.

For $q=Q(p, e_i, e_j, P)$ and $q \neq p$, $e_i \cap e_j = \{q\}$. Let C be the circle centered at q with the radius $d(p', q)$, $\{p_i\} = C \cap e_i$ and $\{p_j\} = C \cap e_j$. Then C is composed of two circular arcs whose endpoints are p_i and p_j, that is, $C = L(p_i, p_j) \cup R(p_i, p_j)$, where $L(p_i, p_j)$ is the circular arc connecting e_i from its left side and $R(p_i, p_j)$ is the other one connecting e_i from its right side. Since $q=Q(p, e_i, e_j, P)$ and $q \neq p$, we have $p' \notin \{p_i, p_j\}$. If $p' \in L(p_i, p_j)$, then p is inside P by noting that P is oriented counterclockwise and the direction from p_i to q agrees with that of P. We now prove that $p' \in L(p_i, p_j)$.

(ii) For p is on the left side of e_i, $A(\triangle p_i q p) > 0$. Suppose $p' \in R(p_i, p_j)$. It follows that p_j is inside the triangle formed by e_i and p. Using the properties of the triangle signed area, we obtain $A(\triangle p_j q p) > 0$, $A(\triangle q p_j p) < 0$. This means that p is on the right side of qp_j, which contradicts the statement that p is on the left side of e_j. Hence it holds that $p' \in L(p_i, p_j)$.

(iii) Since e_j is on the right side of e_i and $e_i \cap e_j = \{q\}$, there exists a triangle formed by e_i and e_j. Suppose $p' \in R(p_i, p_j)$. By hypothesis, p' is inside the triangle so that the vertex q is not the closest point from p' to e_i and e_j. This contradicts the statement that q is the closest point from p' to e_i and e_j due to $q=Q(p, e_i, e_j, P)$. So it holds that $p' \in L(p_i, p_j)$. □

A similar proof holds for the following theorem:

Theorem 3. Let p be a point, P be a simple polygon, and q be a quasi-closest point from p to P. Assume that P is oriented counterclockwise and $p \neq q$. Then p is outside P if one of the following conditions holds:

(i) p is on the right side of e_i for $q=Q(p, e_i, P)$.
(ii) p is on the right side of both e_i and e_j for $q=Q(p, e_i, e_j, P)$.
(iii) e_j is on the left side of e_i for $q=Q(p, e_i, e_j, P)$.

To use the two theorems, we find first a quasi-closest point. Then, the condition (i) is checked if the quasi-closest point does not coincide with any polygon vertex; otherwise, the conditions (ii) or (iii) is checked sequentially. Due to Theorem 1, the condition (iii) is checked only when the adjacent edges are far from co-linearity. Therefore, the theorems give the criteria for point-in-polygon test, which does not need a tolerance to determine the convexity property of a vertex. Furthermore, the advantage of the criteria is that only the sign of the area is required.

It is worth noting that the two theorems can be applied to the region enclosed by a finite number of simple polygons no matter whether there are holes or not. In such case, the polygons should be oriented so that the interior region always lies on the same side of its edges (see Zalik and Kolingerova, 2001). The proof of Theorem 2 also indicates that for a test point, an identical query result can be obtained from different quasi-closest points. In the following, the process of obtaining the result of point-in-polygon test from a quasi-closest point is simply called an inclusion query, and we will focus on finding a quasi-closest point neighboring to a test point.

4. Finding a quasi-closest point

Let P be a polygon and $R(P)$ be the bounding rectangle of P. Then $R(P)$ can be subdivided into a grid of sub-rectangles with uniform size. In the following, the sub rectangle is called the cell of subdivision.

For a point $p \in R(P)$, we use $Cell(p)$ to denote the cell containing p. Fig. 5 illustrates $Cell(p)$, its eight neighboring cells, and the cells overlapped by the δ-neighborhood of p (which will be simply

called the δ-neighborhood cells). In particular, if a cell contains a subset of points of an edge, we say that the cell is crossed by the edge. For convenience, we use *Cells*(e) to denote the cells crossed by the edge (or line segment) e, and *Edges*(C) to denote the edges crossing the cell C. If a cell is not crossed by any edge, we say that the cell is empty.

Based on the subdivision, a quasi-closest point can be obtained from the following result:

Theorem 4. Let $p \in R(P)$ and $q=Q(p, e_i)$. Then q is a quasi-closest point if q is the closest point from p to the edges crossing the cells *Cells*(pq), that is,

$d(p, q)=\min\{d(p, q') | q' \in e, e \in Edges(C), C \in Cells(pq)\}$.

Proof. This result is trivial if $p=q$. For $p \neq q$, suppose that there exists an edge $e_j (j \neq i)$ such that $pq \cap e_j=\{q^*\}$. It follows that $Cell(q^*) \in Cells(pq)$ due to $q^* \in pq$. Noting the condition, we have $q=Q(p, e_j)$.

(a) If $q \neq q^*$, it follows that $d(p, q^*) < d(p, q)$, contrary to the condition. Hence we obtain $pq \cap e_k=\emptyset$ $(1 \leq k \leq n, k \neq i)$.
(b) If $q=q^*$, it follows that q is the vertex connecting e_i and e_j due to that P is a simple polygon. As a result, $pq \cap e_k=\emptyset$ $(1 \leq k \leq n, k \neq i, j)$.

By the definition of a quasi-closest point, we obtain the result. □

From Theorem 4, an algorithm is developed for obtaining a quasi-closest point; from Theorem 2 and 3, the algorithm of an inclusion query is presented in the following as well:

Algorithm: Finding aquasi-closest point.

Input: a point p, a set of cells *initcells* (*composed by one or several cells neighboring to p*)
Output: a quasi-closest point q

0: *curcells* ← *initcells*
1: Find the closet point of p to the edges crossing *curcells*, say q
2: Set a flag for each of *curcells* to indicate that it has been used
3: If each of *Cells*(pq) has been used, then return q
4: *curcells* ← *Cells*(pq), go back to step 1

Algorithm: Perform an inclusion query (for a user-defined tolerance δ).

Input: a point p, a quasi-closest point q
Output: *IN*, *OUT* or *ON*

1: If $d(p, q) \leq \delta$, return *ON*
2: If $q=Q(p, e_i, P)$ then
3: If p is on the left side of e_i, return *IN*
4: Else if $q=Q(p, e_i, e_j, P)$ then
5: If p is on the left side of both e_i and e_j, return *IN*
6: If p is on the right side of both e_i and e_j, return *OUT*
7: If e_j is on the right side of e_i, return *IN*
8: End If
9: return *OUT*

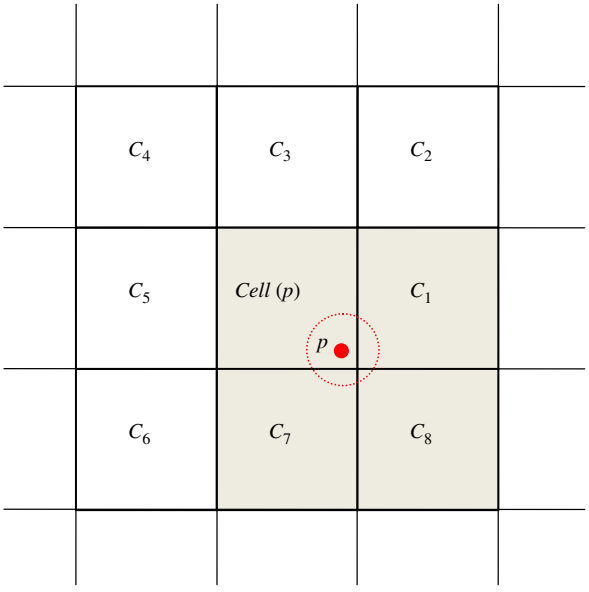

Fig. 5. Illustration for a test point p, the cell *Cell*(p) containing p, and eight neighboring cells of *Cell*(p). Among these cells, cells overlapped by δ-neighborhood of p are indicated in grey color.

Now, we make some remarks on the algorithm for finding a quasi-closest point.

(1) The initial cells of the algorithm are either one of the non-empty neighboring cells or the δ-neighborhood cells for a user-defined tolerance δ. The former is used in the preprocessing stage to determine the inclusion property of an empty cell, and the latter is used in the processing stage to determine the inclusion property of a test point. Note that the latter is used only when some of the δ-neighborhood cells are non-empty. As a result, for a test point p and the quasi-closest point q, q is the closest point of p to all the edges if $d(p, q) \leq \delta$. From the distance $d(p, q)$, the algorithm of an inclusion query can resolve the singularity that a test point is on an edge.
(2) The algorithm really returns a quasi-closest point and its corresponding edge(s), which will be used for the inclusion query.
(3) The loop of the algorithm would terminate in a finite number of steps because the initial cells are composed of one or several of the neighboring cells.
(4) The average-case time complexity depends on the average number of the edges crossing a non-empty cell. In the real applications of GIS, the average number is almost a constant independent of the number of the polygon edges. Thus, the time complexity can be regarded as a constant.

Combining the two algorithms, we obtain a new method for point-in-polygon test based on a quasi-closest point. For convenience, the method is named QCPM (abbreviated for Quasi-Closest Point Method). QCPM requires two parameters, that is, a test point and initial cells; its result will indicate the location of the point such as on an edge, inside or outside a polygon. In particular, whether a point is on an edge is determined by the minimum distance.

5. Point-in-polygon tests for a large number of points

To accelerate the point-in-polygon tests for a large number of points, we preprocess a polygon using the method of Zalik and Kolingerova (2001). First, the bounding rectangle of a polygon is

subdivided into uniform-sized cells. Let n be the number of the polygon edges, W and H be the width and height of the bounding rectangle, respectively. Then the dimension of a cell is determined by

$$\text{Size}_x = \frac{W}{\text{NoOfCell}_x}, \quad \text{Size}_y = \frac{H}{\text{NoOfCell}_y},$$

where

$$\text{NoOfCell}_x = 2\left\lfloor \frac{W\sqrt{n}}{H} \right\rfloor, \quad \text{NoOfCell}_y = 2\left\lfloor \frac{H\sqrt{n}}{W} \right\rfloor.$$

Next, each edge of the polygon is stored in the cells crossed by its corresponding line segment. The crossed cells can be enumerated by a variant of the traversing algorithm (Bresenham, 1965; Fujimoto and Iwata, 1983; Cleary and Wyvill, 1988; Zalik et al., 1997). In addition, we add a flag variable for each cell to indicate whether it has been used.

Finally, the empty cells are classified as inside or outside a polygon. Here, QCPM is used for the classifications when necessary. As shown in the following algorithm, when an empty cell does not neighbor to any cell already classified or out of the bounding rectangle, QCPM is employed with the center point of the empty cell as its test point and one of the non-empty neighboring cells as its initial cells.

Algorithm: Classify the empty cells for the bounding rectangle $R(P)$.

```
For each cell C of R(P)
  If C is empty
    If one of the eight neighboring cells of C is out of R(P)
      Label C with OUT
    Else If one of the eight neighboring cells of C has been
  labeled
      Label C with the inclusion property of the labeled cell
    Else //some of the neighboring cells of C are non-empty
      Find the quasi-closest point for the center point of C
  with one of non-empty neighboring cell as the initial cells
      Label C with the result obtained by an inclusion query
    End If
  End If
End For
```

Based on the preprocessing result, the inclusion properties of a large number of test points can be determined efficiently. For a test point p and a user-defined tolerance δ, if all the δ-neighborhood cells are empty, the property of p is obtained directly from that of $Cell(p)$; otherwise, the property is determined by QCPM with the δ-neighborhood cells as the initial cells. Therefore, the two classifications, one for an empty cell and the other for a test point, are united in this method.

6. Experiments

The algorithms proposed in this paper have been implemented in Visual C++ language, and the experiments were performed on a PC with 512 M RAM and a Pentium CPU (2.93 GHz).

6.1. Experimental tests

We perform the point-in-polygon tests with 1000×1000 points evenly distributed in the bounding rectangle of polygon(s). As shown in Fig. 6, an artificial polygon is used for the first set of experiments, which give counterexamples to the method proposed by (Zalik and Kolingerova, 2001). The counterexamples come from the two areas emphasized in Fig. 6c, where the closest point found by the algorithm is either outside the cell containing the test point or coincident with a polygon vertex. Fig. 6d shows the results from the new method with a larger user-defined tolerance, and among the 1,000,000 test points, 133,119 points are lying on an edge. The comparative experiment shows that the new method appears to offer stability over a similar method and moreover consistently deals with the case of a test point on an edge.

In the second set of experiments, we investigate the performance of the new method. Fig. 7 shows the polygons presented by (Zalik and Kolingerova, 2001). The running time of the point-in-polygon tests performed on these polygons is listed in Table 2. Besides, Table 2 also lists the two factors that affect the performance of the new algorithm, where λ is the ratio of the non-empty cells to all the cells, and κ is the average number of the edges stored in a non-empty cell. The same experiments are also performed on the artificial polygons (see Fig. 8 and Table 3), which are randomly generated in the method described by Li et al. (2007). From the data in Tables 2 and 3, we observe that for the edge number n, as n goes larger in Table 2, λ becomes smaller and the processing time lessens; as n goes larger in Table 3, κ becomes larger and the running time increases. Obviously, the data in Table 2 simulate the real applications whereas the data in Table 3 hardly arise in reality. Anyway, these experiments confirm that the factor κ can be regarded as a constant independent of n, and on average the preprocessing time is linear with n. In particular, the processing time is a constant independent of n, that is, an inclusion test requires constant time.

In the final set of experiments, we compare the running time of different algorithms. Table 4 lists the running time of two algorithms, where Zalik's algorithm comes from (Zalik and Kolingerova, 2001) and its program is downloaded from the website http://www.iamg.org/index.php? option=com_content&task=view&id=175&Itemid=165. Note that the cell number of rows or columns is limited to 100. Although the new algorithm runs faster, we believe that the difference comes from the different implementations. Nevertheless, the results show that the efficiency of the new algorithm is comparable with that of a similar algorithm.

6.2. Case study

The new method can be applied to point-in-polygon overlay in GIS. The point-in-polygon overlay is a spatial operation in which points from one feature dataset are overlaid on the polygons of another to determine which points are contained within the polygons (see Fig. 9).

The case study uses two shape files. The shape file for polygons includes 23266 edges as well as 61 polygons (two of them are holes), and the shape file for spots includes 1377 points. The aim of the study is to test the capability of the method to identify which region contains a test point in a map. The overall workflow is illustrated as follows:

1. Read the map from a shape file.
2. Form the administrative boundaries into oriented polygons.
3. Preprocess the map by subdividing its bounding rectangle into grid cells, recording the borderlines into the cells, and identifying the region to which each empty cell belongs.
4. Read the spots from a shape file.
5. Identify the region to which each spot belongs.

At step 2, it is straightforward to represent the administrative regions in oriented polygon because the polygons in shape file dataset are already oriented. However, the polygons of a map are

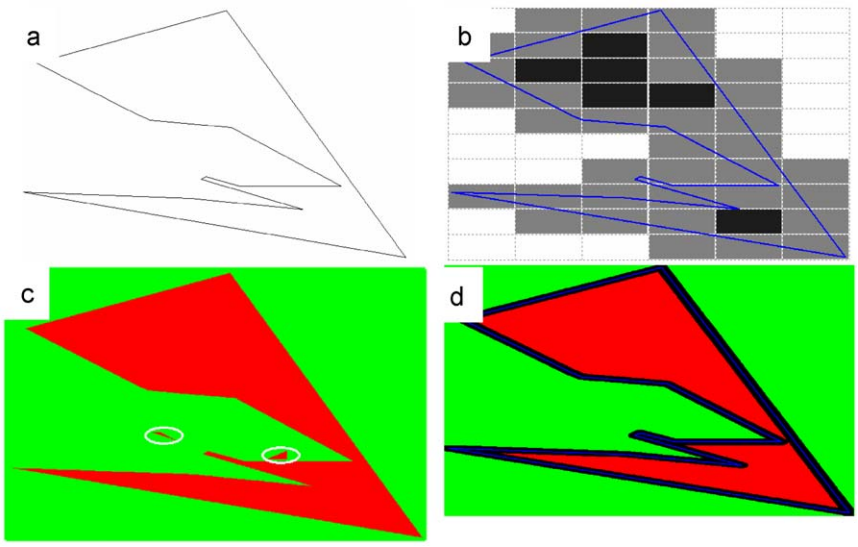

Fig. 6. Point-in-polygon test results (Red-inside, Green-outside, Black-on edge) (a) an artificial polygon; (b) cells of uniform subdivision; (c) test results from (Zalik and Kolingerova, 2001); (d) test results from new method for a larger user-defined tolerance.

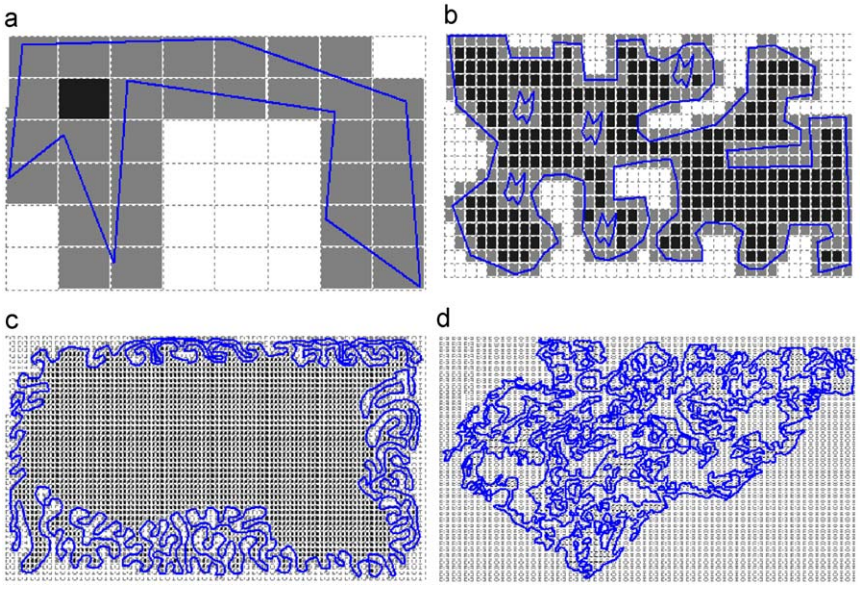

Fig. 7. Polygons and the subdivision from (Zalik and Kolingerova, 2001).

Table 2
Running time on polygons in Fig. 7.

Polygon	Edge number	Cell number	λ	κ	Running time (s)	
					Preprocess	Process
a	10	48	0.67	1.375	0.000044	0. 972392
b	149	684	0.42	1.575	0.000414	0. 801133
c	1248	5060	0.31	1.921	0.003410	0.722073
d	28012	112,992	0.12	3.097	0.054073	0.581358

generally connected, so we have to merge the coincident edges such that the two sides of the merged edge belong to different regions. In Fig. 9, the merged edges are indicated in blue color, and the edges in red color indicate that one of its sides is out of the whole country. Especially, two numbers are assigned to each edge for identifying the regions on the left and right sides.

Now, we revise the algorithm for an inclusion query so that the revised algorithm can return a number for identifying the region on the left or right side. For a quasi-closest point such as $Q(p, e_i, P)$ or $Q(p, e_i, e_j, P)$, the identification number can be obtained directly from the edge e_i. If a quasi-closest point is coincident with a vertex connecting more than two edges, we have to select two edges for the inclusion query. In Fig. 10, for example, we select the two edges adjacent to Region 1 if p is a study spot. For the study spot p and its quasi-closest point q, the selection can be accomplished by evaluating the angles between pq and the connected edges. Thus, we obtain the revised method for identifying the polygon to which a test point belongs.

At step 3, the bounding rectangle of the map is subdivided into a group of grid cells, and each borderline of the map is recorded

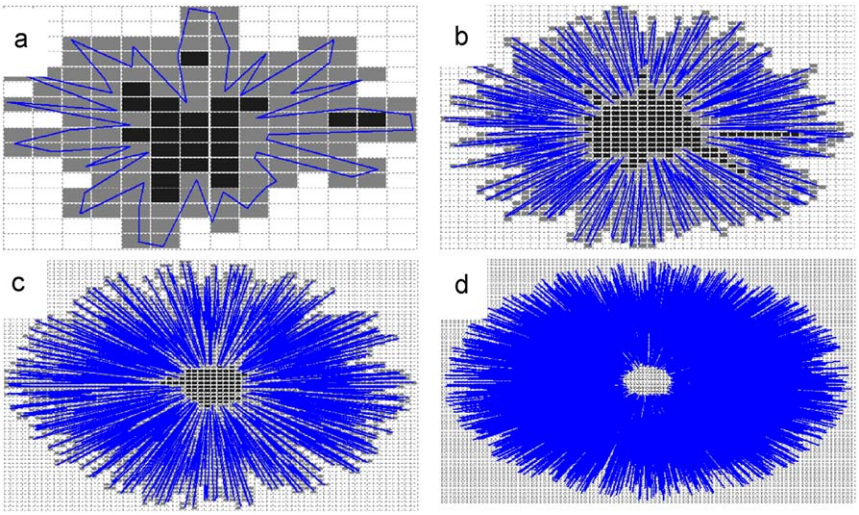

Fig. 8. Polygons randomly generated.

Table 3
Running time of polygons in Fig. 8.

Polygon	Edge number	Cell number	λ	κ	Running time (s)	
					Preprocess	Process
a	50	224	0.50	1.770	0.000203	0.888035
b	500	2116	0.58	3.423	0.002139	1.157722
c	1001	4224	0.65	4.683	0.005793	1.412953
d	5000	20448	0.68	9.856	0.049777	2.145732

Table 4
Running time of different algorithms (in seconds).

Polygon(s) in Fig. 7	Edge number	Cells of subdivision	κ	Zalik's algorithm		The new algorithm	
				Preprocess	Process	Preprocess	Process
a	10	6×8	1.375	0.000055	**2.555309**	0.000044	**0.972392**
b	149	18×38	1.575	0.000645	**1.839879**	0.000414	**0.801133**
c	1248	46×100	1.988	0.004844	**1.707092**	0.002827	**0.740552**
d	28,012	100×100	9.200	0.042282	**7.483888**	0.024614	**1.205676**

into the cells crossed by the borderline. Then, the revised method is used to identify the region to which an empty cell belongs, and correspondingly, an identification number of the region is stored in each empty cell.

At step 5, the revised method is also employed to identify the region to which a spot belongs when necessary. The results are visualized in Fig. 9, where the spots in different regions are indicated by different colors. For the case with 1377 spots and 61 regions, the revised method is used 1377 times. Other methods, such as the ray crossing number method or the sum of angles method, can answer the point-in-polygon problem, but cannot directly answer such problems as to which polygon a point belongs. Using these methods, we can identify the polygon by point-in-polygon test for each polygon of the map; however, for the overlay operation, such tests need to be performed $1377 \times 61/2$ times on average.

7. Conclusion

This paper has presented a robust solution to the point-in-polygon problem. The solution employs a quasi-closest point that can be locally found by the uniform subdivision cells. Based on the quasi-closest point, an inclusion query obtains its result from the sign of triangle signed area, and avoids using a dubious tolerance to identify the convexity property of a polygon vertex. Although obtaining a quasi-closest point requires a little more time than finding the closest point from an individual cell, the extra time cost is rewarded with the correct answer to the point-in-polygon problem as well as the consistent treatment of the singular case that a test point is on an edge.

In this paper, the two classifications (one in the preprocessing stage and the other in the processing stage) are achieved by the inclusion query based on a quasi-closest point. This simplifies the

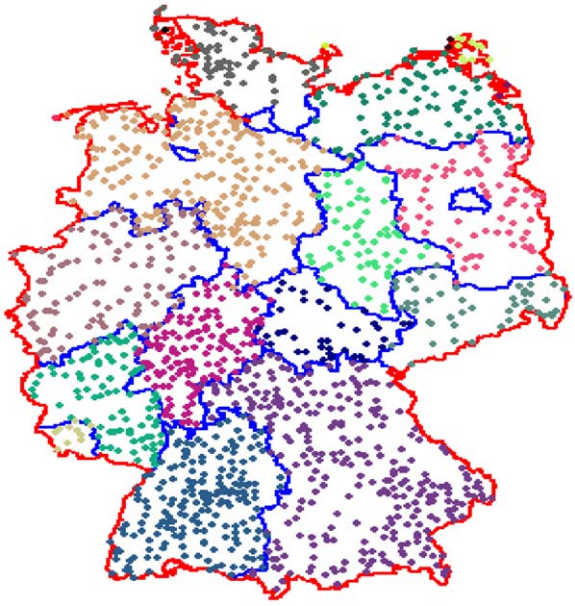

Fig. 9. Point-in-polygon overlay on map of Germany.

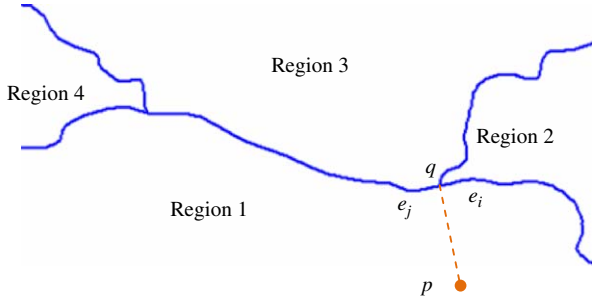

Fig. 10. Quasi-closest point connecting more than two edges.

task of programming and distinguishes from the majority of other approaches.

The present study demonstrates the feasibility of the point-in-polygon test based on a quasi-closest point. In the future, we will apply the method to an inclusion test for the planar region enclosed by the curves such as circular arcs and free-form curves.

Acknowledgements

The research was supported by Chinese 973 Program (2004CB719400), the National Science Foundations of China (60625202, 90715043) and Chinese 863 Program (2007AA-040401). The second author was supported by the project sponsored by a Foundation for the Author of National Excellent Doctoral Dissertation of PR China (200342) and the Fok Ying Tung Education Foundation (111070). In addition, the authors would like to thank the reviewers for their helpful comments.

References

Bresenham, J.E., 1965. Algorithm for computer control of a digital plotter. IBM Systems Journal 4 (1), 25–30.
Cleary, J.G., Wyvill, G., 1988. Analysis of an algorithm for fast ray tracing using uniform space subdivision. The Visual Computer 4 (2), 65–83.
Ellis, F., 2001. Vector Overlay Processing—Specific Theory. GIS Self Learning Tool—University of Melbourne, Department of Geomatics, Melbourne, Australia. 〈http://www.sli.unimelb.edu.au/gisweb/VOModule/VO_Theory.pdf〉 [last access: 22 Oct, 2009].
Etzion, M., Rappoport, A., 1997. On compatible star decomposition of simple polygons. IEEE Transactions on Visualization and Computer Graphics 3 (1), 87–95.
Feito, F.R., Torres, J.C., Urena, A., 1995. Orientation, simplicity, and inclusion test for planar polygons. Computers & Graphics 19 (4), 595–600.
Fujimoto, A., Iwata, K., 1983. Jag-free images on raster displays. IEEE Computer Graphics and Applications 3 (9), 26–34.
Fujimoto, A., Tanaka, T., Iwata, K., 1986. ARTS: accelerated ray-tracing system. IEEE Computer Graphics and Applications 6 (4), 16–26.
Gombosi, M., Zalik, B., 2005. Point-in-polygon tests for geometric buffers. Computers & Geosciences 31 (10), 1201–1212.
Hormann, K., Agathos, A., 2001. The point in polygon problem for arbitrary polygons. Computational Geometry 20 (3), 131–144.
Huang, C.W., Shih, T.Y., 1997. On the complexity of point-in-polygon algorithms. Computers & Geosciences 23 (1), 109–118.
Jiménez, J.J., Feito, F.R., Segura, R.J., 2009. A new hierarchical triangle-based point-in-polygon data structure. Computers & Geosciences 35 (9), 1843–1853, doi:10.1016/j.cageo.2008.09.013.
Li, J., Wang, W.C., Wu, E.H., 2007. Point-in-polygon tests by convex decomposition. Computers & Graphics 31 (4), 636–648.
Martinez, F., Rueda, A.J., Feito, F.R., 2006. The multi-L-REP decomposition and its application to a point-in-polygon inclusion test. Computers & Graphics 30 (6), 947–958.
Nordbeck, S., Rystedt, B., 1967. Computer cartography point-in-polygon programs. BIT Numerical Mathematics 7 (1), 39–64.
Preparata, F.P., Shamos, M.I., 1991. In: Computational Geometry: An Introduction. Springer, New York 390 pp.
O'Rourke, J., 2000. In: Computational Geometry in C. Cambridge University Press, Cambridge 376 pp.
Salomon, K.B., 1978. An efficient point-in-polygon algorithm. Computers & Geosciences 4 (2), 173–178.
Taylor, G., 1994. Point in polygon test. Survey Review 32 (254), 479–484.
Wang, W.C., Li, J., Wu, E.H., 2005. 2D point-in-polygon test by classifying edges into layers. Computers & Graphics 29 (3), 427–439.
Zalik, B., Clapworthy, G., Oblonsek, C., 1997. An efficient code-based voxel-traversing algorithm. Computer Graphics Forum 16 (2), 119–128.
Zalik, B., Kolingerova, I., 2001. A cell-based point-in-polygon algorithm suitable for large sets of points. Computers & Geosciences 27 (10), 1135–1145.